First Hit Fwd Refs

Generate Collection Print

L6: Entry 2 of 11 File: USPT Mar 12, 2002

DOCUMENT-IDENTIFIER: US 6356887 B1
See image for Certificate of Correction
TITLE: Auto-parameterization of database queries

Abstract Text (1):
An auto-parameterization process transforms a database query into a parameterized basic query form by replacing any constant values in the query with parameters. The auto-parameterization process attempts to generate a safe execution plan from the basic query form if there is currently no such plan available. A safe execution plan is defined as an execution plan that is optimal over a range of values for the parameters. If a safe execution plan can be generated, it is passed for execution, along with the constant values that were present in the query. If a safe execution plan cannot be generated, the auto-parameterization process passes a specific execution plan for execution. The safe execution plan is cached either at the time it is created or at the time it is executed. The cache is searched each time a parameterized basic query plan is generated by the auto-parameterization process. The auto-parameterization process also evaluates the query before creating the corresponding parameterized basic query form to determine if it is likely that a safe execution plan can be generated for the query.

Brief Summary Text (6):
SQL query compilation in database systems can take significant time when compared to the execution time of some simple queries. For example, the execution time to lookup a row in an index and modify some of its fields can be considerably smaller than the time it takes to parse, normalize, optimize, and prepare the corresponding SQL statement into the particular set of sub-queries that perform the required operations with the least processing cost (commonly referred to as an "execution plan"). When many simple requests are issued to a database, compilation time consumes a major portion of system resources.

Brief Summary Text (7):
A common approach to address this problem has been through the use of stored procedures. A stored procedure contains one of more SQL statements that are compiled once into their respective execution plans and stored in main memory, so they can be executed multiple times without having to be recompiled. Parameters in the stored procedure provide flexibility to this approach. For example, a stored procedure can do the database work needed to transfer money between two accounts. Such a procedure requires touching a few rows, perhaps in different tables, and it is parameterized off account numbers and amount. After the stored procedure is compiled and stored in memory, there is no need to re-compile it for later executions. Stored procedures are a standard feature in current database systems.

Brief Summary Text (8):
When applications use dynamic SQL, as opposed to stored procedures, database systems typically need to compile each statement, then execute. Many applications currently use dynamic SQL, as opposed to stored procedures, for several reasons. Some of them are legacy code; others did not want to manage the overhead of stored procedures, which are persistent objects and as such require some amount of administration. When those applications issue a large number of simple statements, compilation time can become the dominant cost factor.

Brief Summary Text (12):
An auto-parameterization process transforms a database query that is input to a database server into a parameterized basic query form by replacing any constant values in the query with parameters. The auto-parameterization process attempts to generate a safe execution plan from the basic query form if there is currently no such plan available. A safe execution plan is defined as an execution plan that is optimal over a range of values for the parameters. If a safe execution plan can be generated, it is passed to the execution stage of the database server, along with the constant values that were present in the query. If a safe execution plan cannot be generated, the auto-parameterization process passes an execution plan that contains the specific parameters to the execution stage. The safe execution plan is cached either at the time it is created or at the time it is executed. The cache is searched each time a parameterized basic query plan is generated by the auto-parameterization process.

Brief Summary Text (13):
In one aspect of the invention, the auto-parameterization process analyzes the query before creating the corresponding parameterized basic query form to determine if it is likely that a safe execution plan can be generated for the query.

Drawing Description Text (5):
FIGS. 4A and 4B are diagrams of a query plan data structure and an execution context data structure, respectively, for use in an exemplary implementation of the invention.

Detailed Description Text (5):
FIG. 1 is a diagram of the hardware and operating environment in conjunction with which embodiments of the invention may be practiced. The description of FIG. 1 is intended to provide a brief, general description of suitable computer hardware and a suitable computing environment in conjunction with which the invention may be implemented. Although not required, the invention is described in the general context of computer-executable instructions, such as program modules, being executed by a computer, such as a personal computer. Generally, program modules include routines, programs, objects, components, data structures, etc., that perform particular tasks or implement particular abstract data types.

Detailed Description Text (15):
A system level overview of the operation of an exemplary embodiment of the auto-parameterization of the present invention is described by reference to FIG. 2. FIG. 2 illustrates a database system 200 that incorporates an auto-parameterization subsystem 201. The database system 200 executes on a computer such as local computer 20 or remote computer 49 that is acting as a database server. The auto-parameterization subsystem 201 is made up of four components: an auto-parameterization driver 202, a plan, or procedure, cache 203, a syntax analysis module 204, and a safety analysis module 205. The database system 200 prepares and executes queries using a parser 206, an optimizer (or translation module) 207, and an execution engine 208, as is common practice in the art.

Detailed Description Text (19):
Thus, a single execution plan for the basic query form can execute any of its variants, when values are provided to a set of parameters.

Detailed Description Text (20):
Queries Q1 and Q2 are input into the parser 206 in the normal fashion. The syntax analysis module 204 analyzes each query to determine if it is a good candidate for auto-parameterization, i.e., it is likely that a "safe" execution plan can be generated for the query as will be discussed next. In the example shown in FIG. 2, query Q1 is not considered likely to be safe and so it follows the normal preparation path as illustrated by actions 210 and 211. In an alternate embodiment

not illustrated, the syntax analysis 204 is not present in the parser 206 and all queries are subjected to the safety analysis described next.

Detailed Description Text (21):
In the exemplary database system shown in FIG. 2A, Q2 is considered likely to be safe by the syntax analysis module 204, so a sequence or query tree, tree(Q2), representation of query Q2 is created by the parser 206 and sent to the auto-parameterization driver 202 (action 212). The auto-parameterization driver 202 replaces any constant values P in the query Q2 with parameters to create a parameterized basic query form Q2'. The auto-parameterization driver 202 creates a name or identifier, name(Q2'), for the basic query form Q2', and uses name(Q2') to search the plan cache 203 for an existing execution plan for the basic query form Q2' (action 213). If it exists, a matching cached execution plan, plan(Q2'), is retrieved (action 214) and passed, along with the constant values P, to the execution engine 208 for execution (action 215).

Detailed Description Text (22):
If there is no plan(Q2') in the cache 203, the auto-parameterization driver 202 passes a query tree, tree(Q2'), for the basic query form Q2' to the optimizer 207, along with the constant values P (action 216). The optimizer 207 creates an optimized execution plan for the query Q2 in its normal fashion. The safety analysis module 205 determines whether the optimized plan for query Q2 will be the optimal plan for a range of values for the parameters, i.e., it is considered "safe" for those values. The actual constant values P are used by the safety analysis module 205 as typical values for the parameters.

Detailed Description Text (23):
If the optimized plan for Q2 is deemed safe by the safety analysis module 205, the optimizer 207 returns the safe execution plan, plan(Q2'), and a safety flag to the auto-parameterization driver 202 (action 219). The safety flag causes the auto-parameterization driver 202 to store the plan(Q2') in the plan cache 203 under the unique identifier, name(Q2') (action 220). The auto-parameterization driver 202 also sends the plan(Q2') and the constant values P to the execution engine 208 for execution (action 215).

Detailed Description Text (24):
If, however, the plan(Q2') is not considered safe, the optimizer 207 generates a specific, non-parameterized execution plan, plan(Q2), for the query Q2, just as it would had query Q2 arrived through the normal preparation path 210. The plan(Q2) is passed back to the auto-parameterization driver 202 (action 217), which then submits the plan(Q2) to the execution engine 208 (action 218).

Detailed Description Text (25):
The system level overview of the operation of an exemplary embodiment of the invention has been described in this section of the detailed description. The constant values in a query are replaced with parameters to form a parameterized basic execution plan for the query. A plan cache is searched to determine if the basic execution plan exists. If the basic execution plan exists in the cache, it has been deemed the optimal execution plan for any values of the parameters and is considered safe. The parameters in the basic execution plan are replaced with the corresponding constant values and the resulting query is executed. If no matching basic execution plan is found in the cache, an attempt is made to compile a safe execution plan. If a safe execution plan can be created, it is stored in the cache. If not, a specific execution plan is executed. While the invention is not limited to any particular database system, for sake of clarity a simplified database system having a parser, an optimizer and an execution component has been described.

Detailed Description Text (27):
In the previous section, a system level overview of the operation of an exemplary embodiment of the invention was described. In this section, the particular methods

performed by a computer executing such an exemplary embodiment when acting as a database server are described by reference to a series of flowcharts. The methods to be performed by the computer constitute computer programs made up of computer-executable instructions. Describing the methods by reference to a flowchart enables one skilled in the art to develop such programs including such instructions to carry out the methods on suitable computers (the processor of the clients executing the instructions from computer-readable media). FIGS. 3A, 3B, 3C, and 3D illustrate the methods to be performed by a computer according to the exemplary embodiment of the auto-parameterization subsystem shown in FIG. 2.

Detailed Description Text (29):
In FIG. 3B, when the auto-parameterization process receives the query tree from the syntax analysis, it separates the query tree into its basic query form and replaces all constant values by parameters (block 311). A unique identifier is created for the basic query form (block 313). The plan cache is searched using the identifier (block 315). In one exemplary embodiment, the unique identifier is a normalized form of the SQL text of the basic query form, where all redundant spaces are removed and the text is all upper case. A hash table is used to speed up the search, with hash values computed from keywords, identifiers, and parameters that compose the query. If a corresponding plan is found, it is passed to an execution process along with the values of the parameters for the query (block 317). The execution process is described in detail below in conjunction with FIG. 3D.

Detailed Description Text (30):
If no matching execution plan is cached, the auto-parameterization process invokes an optimization process, as discussed in detail in conjunction with FIG. 3C, to create an execution plan (block 319). The optimization process returns an execution plan. If a safety analysis process within the optimization process has determined that the execution plan is safe, the optimization process also returns a safety flag. The safety flag indicates the query is considered safe and thus is auto-parameterized. The presence of the safety flag is tested by the auto-parameterization process at block 321. If there is no safety flag, the execution plan that was created by the optimization process is specific to the query and is immediately passed to the execution process (block 323). A safe execution plan is stored in the cache under the unique query name (block 325) and is then passed to the execution process at block 317, along with the parameter values. In an alternate embodiment not illustrated, a safe execution plan is not cached until it is executed due to the possibility that changes to the schema of the underlying database can render the plan unsafe before it is executed. Examples of when changes in the database render a plan unsafe are provided further below and in the next section.

Detailed Description Text (31):
Turning now to FIG. 3C, an exemplary embodiment of a safety analysis method is described that operates in conjunction with standard optimization processing to compile queries for execution. The safety analysis method operates to perform the functions described in the previous section with reference to the safety analysis module 205 in FIG. 2. When the optimization process receives the basic query form from the auto-parameterization process, it creates an optimized plan for the basic query form (block 331). The optimization process uses the values of the parameters as hints in a well-known technique commonly referred to as "parameter sniffing." The optimized plan is then analyzed for safety (block 333). The details of the safety analysis for one exemplary embodiment is discussed next.

Detailed Description Text (34):
When there are more alternatives, for example two filter conditions C1, C2 that can be used on different indices I1, I2, respectively, then there are several alternative execution plans available. When filter conditions contain only equalities, and columns compared have uniform, independent distributions, a plan can be considered safe.

Detailed Description Text (35):
Queries that contain multiple tables also have a number of possible execution plans. Again, safe parameterization can be done when conditions are equalities, and statistics are uniform and independent.

Detailed Description Text (36):
The execution plan for a basic form can be made more elaborate, with plan alternatives tied together by means of "switch" operators that choose between those alternatives at run-time, depending on actual parameter values to ensure the safety of the plan for different parameter values.

Detailed Description Text (37):
If the execution plan is deemed safe (block 335), a safety flag is passed back to the auto-parameterization process with the parameterized execution plan (block 337). If the execution plan is not deemed safe, the actual values are substituted for the corresponding parameters (block 339). Additional query analysis (block 341) can be performed on the resulting, non-parameterized execution plan before is returned (block 343). The additional analysis can include necessary constraint checks and interval merging, for example.

Detailed Description Text (38):
Check constraints can be used in a database to enforce particular conditions are met by the data stored in the database. When check constraints are declared in the database, they can be combined with selection predicates to infer contradictions. For example, a given table may contain only customers with an address in either CA, OR, or WA. If a query includes a condition of the form state=`MA` then the check constraint indicates that the output is empty, at compile time, without having to touch the table at execution time. However, after parameterization, some compile-time contradiction detection can no longer be performed as part of the optimization process and is moved to the execution process. But when a specific execution plan is generated, the value of the parameters are known so a contradiction with check constraints can be detected.

Detailed Description Text (39):
Similarly, interval merging cannot be accomplished without actual values and is also moved into the execution process unless the execution plan is specific. For example, if a query contains conditions A>@X and A>@Y, then a single lower bound MIN@X,@Y) must be computed before the query is executed.

Detailed Description Text (40):
A cached safe execution plan can become unsafe due to changes in the underlying database and therefore must be deleted from the cache. Additionally, some aging process can be applied to the cached plans to delete less-used plans from the cache. FIG. 3D illustrates a method of maintaining the cache that can be invoked by the auto-parameterization process upon searching the cache or by the execution process upon executing a query so that unsafe plans can be deleted. If the underlying database is changed (block 351), the process evaluates each plan in the cache to determine if it is impacted by the change (block 353). Possible reasons for a plan being impacted include a significant change in the statistics used to derive the plan, or a change in the schema of the tables used in the query. One exemplary embodiment of the auto-parameterization process relies on database statistics collected as described in pending U.S. patent application Ser. No. 09/213087 (entitled "Automatic Database Statistics Creation") in determining the need to re-optimize a plan as described in pending U.S. patent application Ser. No. 09/212933 (entitled "Automatic Database Statistics Maintenance And Plan Regeneration"), both assigned to the assignee of the present application.

Detailed Description Text (41):
If the schema of the table has changed, the plan must be re-optimized because the

compiled plan stores information about a particular version of the table. A new schema version means that the information stored in the compiled plan is now invalidated. If a plan must be recompiled, the plan is deleted from the cache (block 355). The optimization process is then invoked to see if the safety analysis can find a safe plan (block 357). If the optimization process returns a safe plan (block 359), it is cached (block 360).

Detailed Description Text (201):
An SQL Server execution plan is made up of a query plan and an execution context. The query plan is a reentrant, read-only data structure 400 that can be used by any number of users as illustrated in FIG. 4A. Once created, multiple users are able to use the same query plan. An execution context data structure 410 as illustrated in FIG. 4B is created for each user 411 that is currently executing the query represented by a query plan data structure 400. (pointer 412) to hold data 415, such as parameter values, that are specific to that user's execution.

Detailed Description Text (207):
If the cache lookup for a compiled query plan fails, QPO is invoked to generate an execution plan. The constants that were converted to parameters are passed in as a "typical values" hint to QPO to enable parameter sniffing, as used for stored procedures. If QPO generates a safe execution plan it is cached at the end of the compile. An unsafe execution plan cannot be shared and to avoid destroying it, and wasting the work done so far, the unsafe execution plan is attached to the compile context of the containing statement. Such an unsafe execution plan will only be reused if an identical statement is received.

Detailed Description Text (209):
As described in the previous section, certain changes in a database can cause an execution plan to be considered unsafe. SQL Server marks a cached execution plan for a query as unsafe under the following conditions:

Detailed Description Text (212):
Dropping an index used by the execution plan.

Detailed Description Text (213):
An explicit call to recompile the query (WITH RECOMPILE or sp_recompile).

Detailed Description Text (216):
Also as described previously, SQL Server performs an aging process on cached plans. Each query plan and execution context has an associated cost factor 404, 413 that indicates how expensive the corresponding data structure is to compile. These data structures also have an age field 405, 414. Each time the data structure is referenced by a connection, the age field is incremented by the compilation cost factor. For example, if a query plan has a cost factor of 8 and is referenced twice, its age becomes 16. A lazywriter process periodically scans the list of data structures in the plan cache. On each scan, the lazywriter decrements the age field for each data structure by 1. Thus, the age of the present sample query plan is decremented to 0 after 16 scans of the plan cache, unless another user references the plan.

Detailed Description Text (222):
The specific syntax and safety analysis employed by the SQL Server in implementing the auto-parameterization process has been described in this section. In addition, the data structures that comprise an execution plan, and the cache search methodology have also been discussed. Finally, necessary maintenance for the cache has been disclosed.

Detailed Description Text (224):
An auto-parameterization process for use within a database server has been described that transforms a database query into a parameterized basic query form

and attempts to create a safe execution plan from the basic query form, i.e., an execution plan that is optimal over a range of values for the parameters. A safe execution plan is cached for reuse. Although specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement which is calculated to achieve the same purpose may be substituted for the specific embodiments shown. This application is intended to cover any adaptations or variations of the present invention.

Detailed Description Text (225):
For example, those of ordinary skill within the art will appreciate that the processes of the auto-parameterization can be performed in other components of the database server than those shown in the exemplary embodiments of the present application. Furthermore, those of ordinary skill within the art will appreciate that the criteria used to determine the safety of an execution plan is dependent upon the underlying database management system.

Detailed Description Text (226):
The terminology used in this application with respect to SQL relational databases is meant to include all database environments that compute execution plans for database queries. Therefore, it is manifestly intended that this invention be limited only by the following claims and equivalents thereof.

Current US Original Classification (1):
707/2

Current US Cross Reference Classification (1):
707/4

CLAIMS:

1. A computerized method for automatically parameterizng a database query comprising:

creating a basic query form by an auto-parameterization system for the query by replacing any constant values with parameters;

attempting to create a safe execution plan for the basic query form if no safe execution plan by an auto-parameterization system currently exists; and

passing the safe execution plan and the constant values by an auto-parameterization system onto execution if the safe execution plan exists.

2. The method of claim 1, further comprising:

determining if the query is likely to have a safe execution plan; and

creating the basic query form if the query is determined to likely to have a safe execution plan.

3. The method of claim 1, wherein determining if the query is likely to have a safe execution plan comprises:

performing syntax analysis on the query.

4. The method of claim 1, comprising:

creating a unique name for the basic query form; and

storing the safe execution plan in a cache under the unique name.

5. The method of claim 4, wherein the safe execution plan is stored in the cache when it is created.

6. The method of claim 4, wherein the safe execution plan is stored in the cache when it is executed.

8. The method of claim 4, further comprising:

determining if a current safe execution plan exists by searching the cache using the unique name for the basic query form.

9. The method of claim 4, further comprising:

searching the cache using the unique name for the basic query form before executing the safe execution plan; and

creating a specific execution plan if the safe execution plan does not exist in the cache.

10. The method of claim 4, further comprising:

removing the safe execution plan from the cache when it is no longer optimal over a range of values for the parameters.

11. The method of claim 4, further comprising:

removing the safe execution plan from the cache when an age indicator associated with the safe execution plan satisfied pre-determined criteria.

12. The method of claim 1, wherein attempting to create the safe execution plan comprises:

generating an optimal execution plan for the basic query form;

analyzing the optimal execution plan against a range of values for the parameters; and

designating the optimal execution plan as the safe execution plan if the evaluation shows it is optimal over the range of values.

14. The method of claim 1, wherein the execution plan comprises:

a query plan; and

an execution context.

16. The method of claim 1, further comprising passing a specific execution plan onto execution if the safe execution plan does not exist.

17. A computer-readable medium having computer-executable modules stored thereon, the modules comprising:

a syntax analysis module that determines if query is likely to generate a safe execution plan;

an auto-parameterization driver module that creates a parameterized basic query form from the query and further passes any safe execution plan generated from the parameterized basic query form onto an execution module; and

a safety analysis module that evaluates whether the execution plan generated from

the parameterized basic query form is safe.

18. The computer-readable medium of claim 17, further comprising:

a parser module that parses the query into a query tree, submits the query tree to the syntax analysis module for analysis, and further submits the query tree to an optimizer module if the query is not likely to generate a safe execution plan.

19. The computer-readable medium of claim 17, further comprising:

a parser module that parses the query into a query tree, submits the query tree to the syntax analysis module for analysis, and further submits the query tree to a normalizer module if the query is not likely to generate a safe execution plan.

20. The computer-readable medium of claim 19, wherein the normalizer module normalizes the query tree and submits the normalized query tree to the syntax analysis module for analysis, and further submits the query tree to an optimizer module if the query is not likely to generate a safe execution plan.

21. The computer-readable medium of claim 17, further comprising;

an optimizer module that generates the execution plan from the parameterized basic query form.

22. The computer-readable medium of claim 21, wherein the optimizer module generates a specific execution plan for the query if the safety analysis module evaluates the execution plan generated from the parameterized basic query form as unsafe, and wherein the auto-parameterization driver module submits the specific execution plan to the execution module.

23. The computer-readable medium of claim 17, wherein the safety analysis module evaluates the execution plan as safe if the execution plan is optimal over a range of parameter values.

24. The computer-readable medium of claim 17, wherein the auto-parameterization driver module stores any safe execution plan generated from the parameterized basic query form in a cache and searches the cache for the safe execution plan each time it creates the same parameterized basic query form.

25. The computer-readable medium of claim 24, further comprising an execution module that searches the cache for the safe execution plan when it receives the safe execution plan from the auto-parameterization driver module, and further causes a specific execution plan for the query to be generated when the safe execution plan is not found in the cache.

26. The computer-readable medium of claim 17, further comprising an execution module that stores the safe execution plan in a cache when the plan is executed and wherein the auto-parameterization driver searches the cache for the safe execution plan each time it creates the same parameterized basic query form.

27. A computerized system comprising:

a processing unit;

a system memory coupled to the processing unit through a system bus;

a computer-readable medium coupled to the processing unit through a system bus; and

an auto-parameterization sub-system executed from the computer-readable medium by

the processing unit, wherein the auto-parameterization subsystem causes the processing unit to generate a parameterized basic query form from a query, to generate an execution plan from the parameterized basic query form, to evaluate the execution plan for safety, and to submit the execution plan for execution if it is evaluated as safe.

28. The computerized system of claim 27, wherein the auto-parameterization subsystem further causes the processing unit to store the execution plan in the system memory if it is evaluated as safe, and to search the system memory for the execution each time the processing unit generates the parameterized basic query form.

29. The computerized system of claim 28, wherein the auto-parameterized subsystem causes the processing unit to remove the execution plan from the system memory under certain conditions.

30. The computerized system of claim 29, wherein the certain conditions are selected from the group consisting of:

alteration to a schema structure accessed by the query;

modification of statistics used to generate the execution plan; and

satisfaction of pre-determined criteria by an age indicator associated with the execution plan.